Client Side Flow

the client invoke @resourceallocator_2 and then periodically every 2 seconds make a request with a random number of resources. The client is if result returned is not multiplied by 2 it means the the server already blocked this thread request due to the lack of resources.

Function: resourceallocator_2

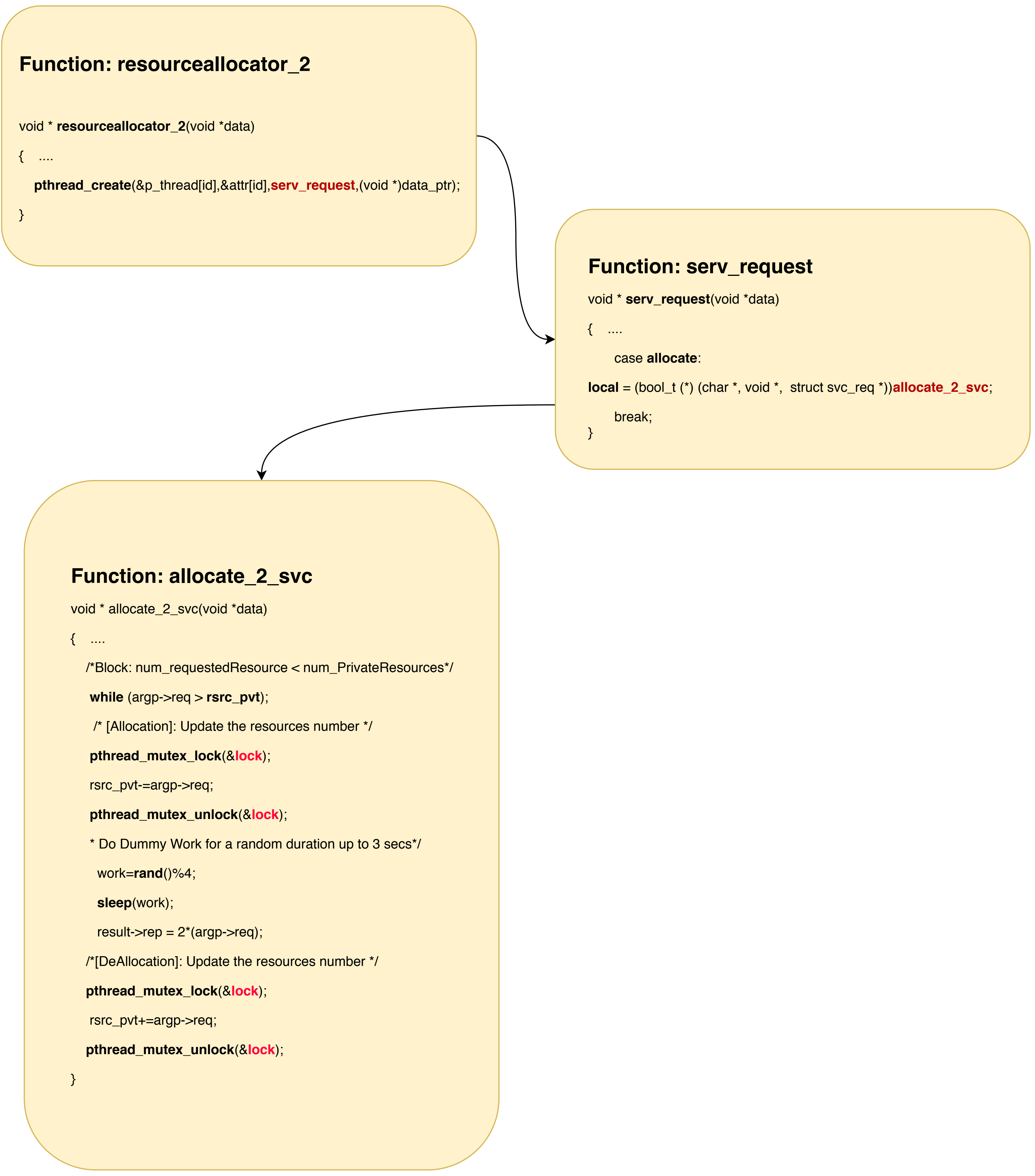
```
void * resourceallocator_2(void *data)
{    ....
      clnt = clnt_create ("127.0.0.1", ResourceAllocator, RA, "udp");
      while(1) {

            allocate_2_arg.req = rand()%10;
            retval_1 = allocate_2(&allocate_2_arg, &result_1, clnt);
            if (result_1.rep != allocate_2_arg.req*2) {
                printf("I'm Blocked\n");
            } else printf("[Result:\t] %ld\n",result_1.rep);

            sleep(2); // Make a request every 2 sec
      }
}
```

Server Side Flow

@resourceallocator_2 is invoked each time for each RPC from any client. Therefore, a thread is created to services this request,



Function: resourceallocator_2

```
void * resourceallocator_2(void *data)
{    ....
   pthread_create(&p_thread[id],&attr[id],serv_request,(void *)data_ptr);
}
```

Function: serv_request

```
void * serv_request(void *data)
{    ....
      case allocate:
local = (bool_t (*) (char *, void *,  struct svc_req *))allocate_2_svc;
      break;
}
```

Function: allocate_2_svc

```
void * allocate_2_svc(void *data)
{    ....
   /*Block: num_requestedResource < num_PrivateResources*/
    while (argp->req > rsrc_pvt);
     /* [Allocation]: Update the resources number */
    pthread_mutex_lock(&lock);
    rsrc_pvt-=argp->req;
    pthread_mutex_unlock(&lock);
    * Do Dummy Work for a random duration up to 3 secs*/
      work=rand()%4;
      sleep(work);
      result->rep = 2*(argp->req);
   /*[DeAllocation]: Update the resources number */
   pthread_mutex_lock(&lock);
    rsrc_pvt+=argp->req;
   pthread_mutex_unlock(&lock);
}
```